Problem-oriented programming

All programmes exist to serve some purposes and to solve some problems. Inherently thus they are problem-oriented. At the same time, no solutions to any problem are perfect. This means that all programmes can be improved. It is therefore important for us to keep our mind on the problems and not on any particular solution.

Problem 1 and what follow is a case study of such problem as we may write our programmes to help answer. Here the subject for our test is this present course we are learning, namely C and Unix.

Problem 1. How to have a fair test?

Problem 1 may be said to be *the* problem in this case. From it spring loads of other problems to consider, both detailed and practical ones. One such problem is how to make a test that is fair to *this* subject.

Since both C and Unix can be considered a skill or a language, we conclude that our attention should be on hands-on experience. But it is one thing to write an answer to a question correctly, and quite another to carry it out in actual practice. Also there is a hazard of answers obtained by asking or telling friends, which could render a test unfair and useless. Keeping all this in mind we choose for our test an oral exam of the 'show-me how-to' type. Then it is important that each person being tested has a whole attention of the one carrying out the test. Therefore we choose for us a sequential test where questions are asked one person after another.

Ideally in such a test one needs to make sure that each person get a different set of questions from others. Also there is an issue of the fairness of the choise of questions in each set. This is impossible since any two questions are different. For now we find our way out of this dilemma by having the questions in the sets randomly chosen.

But randomness as implemented by the library function `rand` in C is in fact merely a very long sequence of integers. Though as a whole the sequence is practically without any patterns, yet it is a known sequence. Which means that if we know where in the sequence we begin we will be able to predict our sequence of numbers by simplying reading off tables. Then our sequence of numbers would not be very random afterall, in the sense that we could find out where it is going next.

But all is not yet hopeless. We still could have our chosen sequence random by choosing our starting point on the `rand` sequence and make sure that *that* is randomly done.

In our case we ensure this randomness in our choice of where in that sequence to begin our sequence by having each student choose an integer which are then added and used as the so-called *seed*, that is our starting point.

All this represents our solution and the way we carry out our test. Algorithm 1 puts together what we have said up to now.

seed $\leftarrow 0$
for each *student* **do**
  choose a number, *num*
  seed $\leftarrow$ *seed* + *num*
endfor
initialise *rand()* with *seed*
choose randomly the order by which students are to be tested
for each *student* **do**
  choose randomly five questions from the set of questions
endfor

Numbers chosen by students are in their usual order 3, 5, 9, 7, 4, 4, 4, 7, 11, 3, 93, 33, 7, 7, 5, 9, 9, 15, 7, 9, 2, 5, 7, 8, 9, 2 and 4. They add up to 288, and this number was used as a seed for our random number generation. Next, the random numbers generated from the seed are put into *id.txt*, which is given in Listing 1.

Listing 1 *Random numbers for students*

```
1 1492921876
2 2031279135
3 1374965837
4 363701165
5 1023699735
6 1225876319
7 961810921
8 2016580502
9 968848298
10 1835762276
11 1490431874
12 197035131
13 89380432
14 1876165494
15 1164775919
16 717716400
17 1636411253
18 135000917
19 239069903
20 1216396086
21 1688011284
22 1256455512
23 1854796271
24 1758355444
25 192848381
26 1049089232
27 722243584
```

The ordering of students extracted from *idd.txt* is 5, 26, 15, 20, 6, 22, 18, 3, 11, 1, 17, 21, 24, 10, 23, 14, 25, 12, 8, 2, 19, 4, 16, 27, 13, 7 and then 9.

Finally, the five questions assigned to each student are shown for all students in Listing 2, which is the contents of *qsts.txt*.

Listing 2 *Five questions to each student*

```
ID Q1 Q2 Q3 Q4 Q5
1 1 20 7 26 13
2 24 65 48 7 44
3 55 36 10 28 45
4 56 1 16 25 53
5 1 37 21 64 63
6 8 16 14 19 40
7 63 21 59 7 46
8 9 32 46 56 38
9 26 47 8 35 12
10 54 26 14 6 50
```

Listing 2 *(continued)*

```
11 1 7 24 21 5
12 23 31 22 37 51
13 64 36 7 57 42
14 54 2 9 34 58
15 48 60 41 56 29
16 54 46 56 2 52
17 43 2 60 1 25
18 64 23 55 61 40
19 21 32 48 14 10
20 37 16 20 5 10
21 3 1 52 60 32
22 41 40 22 42 26
23 64 44 20 1 3
24 21 26 59 43 35
25 65 54 18 13 64
26 54 28 18 60 39
27 22 63 26 16 29
```

Table 1 gives a list of students together with the marks and score from their test.

Number	*Student*				*Marks*		*Total*
1	Fon	2	1	1	1	1	6
2	Mook	2	1	1	4	4	12
3	Eiw	4	4	1	1	1	11
4	Re	2	4	3	4	1	14
5	Meam	2	2	2	1	2	9
6	Oum	1	2	1	2	3	9
7	Jan	4	2	4	4	2	16
8	Film	4	4	2	4	4	18
9	Pear	2	4	2	4	2	14
10	Yo	4	1	2	4	1	12
11	Na	4	2	2	2	4	14
12	Nest	4	4	2	2	2	14
13	Tang	2	4	4	4	4	18
14	Seven	4	1	2	4	4	15
15	Aim	1	2	4	2	3	12
16	AApi	4	1	4	2	1	12
17	AChu	2	1	2	4	2	11
18	Eua	1	4	1	1	1	8
19	Ja	1	2	2	4	4	13
20	Bo	1	1	2	1	1	6
21	Ta	2	4	1	2	1	10
22	Dawn	4	1	1	4	1	11
23	Nulek	1	4	4	4	2	15
24	Bee	2	2	2	4	4	14
25	Namtan	1	4	2	2	1	10
26	Ying	4	1	1	1	1	8
27	Gift	1	1	2	2	1	7

Table 1 `tsmsn`

Note that to compile Table 1 we use `awk` in Unix as follows.

```
awk '{printf "%d %s %d %d %d %d %d %d\n", \
   $1, $3, $4, $5, $6, $7, $8, $4+$5+$6+$7+$8}' ./test1.dat > tmp
```

Here the fields in `test1.dat` are *number*, *name*, *other name*, *score for question 1*, *2*, *3*, *4* and *5*.

The frequency of each question being chosen are in the format *questioin*(*frequency*) in their order, 1(0), 2(7), 3(3), 4(2), 5(0), 6(2), 7(1), 8(5), 9(2), 10(2), 11(3), 12(0), 13(1), 14(2), 15(3), 16(0), 17(4), 18(0), 19(2), 20(1), 21(3), 22(5), 23(3), 24(2), 25(2), 26(2), 27(6), 28(0), 29(2), 30(2), 31(0), 32(1), 33(3), 34(0), 35(1), 36(2), 37(2), 38(3), 39(1), 40(1), 41(3), 42(2), 43(2), 44(2), 45(2), 46(1), 47(3), 48(1), 49(3), 50(0), 51(1), 52(1), 53(2), 54(1), 55(5), 56(2), 57(4), 58(1), 59(1), 60(2), 61(4), 62(1), 63(0), 64(3), and 65(5).

Programme 3 is written to mark Test 1.

Programme 3 *Programme to mark the examination*

```
1  /* Programme to mark Test 1, Kit Tyabandha, 30 Dec 2006 */
2  #include<stdio.h>
3  struct student{
4    int id;
5    char nm[12];
6    char nknm[6];
7    int mrk[10];
8  };
9  void
10 score(){
11   FILE *fpt;
12   int i, j, mrkn=5, studn=27;
13   int scr[studn];
14   struct student stud[studn];
15   for(i=0; i<studn; i++){
16     scr[i] =0;
17   }
18   fpt =fopen("test1.dat", "r");
19   for(i=0; i<studn; i++){
20     fscanf(fpt, "%d %s %s", &stud[i].id, stud[i].nm, stud[i].nknm);
21     for(j=0; j<mrkn; j++){
22       fscanf(fpt, " %d", &stud[i].mrk[j]); fscanf(fpt, "\n");
23     }
24   }
25   fclose(fpt);
26   for(i=0; i<studn; i++){
27     for(j=0; j<mrkn; j++){
28       scr[i] +=stud[i].mrk[j];
29     }
30   }
31   fpt =fopen("score1.opt", "w");
32   fprintf(fpt, "ID\tName\tScore\n\n");
33   for(i=0; i<studn; i++){
34     fprintf(fpt, "%d\t%s\t%d\n", stud[i].id, stud[i].nm, scr[i]);
35   }
36   fclose(fpt);
37 }
38 int
39 main(){
40   score();
41   return 0;
42 }
```

Programme 4 finds frequency of occurrence of questions. The results of this are given above.

Programme 4 *Programme to find frequency of questions*

```
1  /* find frequency of questions, Kit Tyabandha, 9 Jan 2007*/
2  #include<stdio.h>
3  #include<stdlib.h>
4  int
5  main(){
6    char c[2];
7    FILE *fpt;
8    int i, j, q[5], qn=65, id, sn=27;
9    int f[qn];
10   for(i=0; i<qn; i++){
11     f[i]=0;
12   }
13   fpt =fopen("qsts.txt", "r");
14   fscanf(fpt, "%s\t%s\t%s\t%s\t%s\t%s\n", c, c, c, c, c, c);
15   for(i=0; i<sn; i++){
16     fscanf(fpt, "%d\t%d\t%d\t%d\t%d\t%d", \
17       &id, &q[0], &q[1], &q[2], &q[3], &q[4]);
18     for(j=0; j<5; j++){
19       f[q[j]]++;
20     }
21   }
22   fclose(fpt);
23   fpt =fopen("foq.txt", "w");
24   fprintf(fpt, "Question\tFrequency\n\n");
25   for(i=0; i<qn; i++){
26     fprintf(fpt, "%d\t%d\n", i+1, f[i]);
27   }
28   fclose(fpt);
29   exit(0);
30 }
```

Problem and possible solution

As the test was held in a large computer room where students are free to wander around, there is a question of whether those being tested later could learn or gain information from their less unfortunate friends. One possible solution to this is to carry out the test in a separate room and to have students take turns entering it one after another.

This would have an additional advantage that all the necessary materials could then be prepared in advance, for example a header file and a text file for students to demonstrate their solution on.